1. **Histogram Equalization and Stretching**
   Useful URL

   Given an image the code below plots the histogram and cumulative density function of that image. [you will notice that the density of pixel is around the centre of the histogram]

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('wiki.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"
hist,bins = np.histogram(img.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.show()
```

   A good image should have pixel values from all regions of the image, which is the purpose of Histogram equalisation. (code in python using numpy)

```
cdf_m = np.ma.masked_equal(cdf,0)
cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())
cdf = np.ma.filled(cdf_m,0).astype('uint8')
 img2 = cdf[img]
```

we now calculate the histogram and cdf as before

   With Opencv

```
img = cv.imread('wiki.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"
equ = cv.equalizeHist(img)
res = np.hstack((img,equ)) #stacking images side-by-side
cv.imwrite('res.png',res)
```

2. **Image Thresholding**
   Useful URL

   The sample codes in the url basically contain different image thresholding techniques. example:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('gradient.png', cv.IMREAD_GRAYSCALE)
```

```
assert img is not None, "file could not be read, check with
os.path.exists()"
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
titles = ['Original
Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in range(6):
plt.subplot(2,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)
plt.title(titles[i])
plt.xticks([]),plt.yticks([])
plt.show()
```

3. **Image Enhancement with Sobel, LoG, Laplacian filters**
   Useful URL

   We can easily obtain [from OpenCV library] a set of differential filters including Sobel and Laplace filters. Sample code:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('dave.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"
laplacian = cv.Laplacian(img,cv.CV_64F)
sobelx = cv.Sobel(img,cv.CV_64F,1,0,ksize=5)
sobely = cv.Sobel(img,cv.CV_64F,0,1,ksize=5)
plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.show()
```

   LoG [Laplacian of Gaussian]

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
# Apply Gaussian Blur
sigma = 1.0
blurred_image = cv2.GaussianBlur(image, (5, 5), sigma)
# Apply Laplacian
laplacian_image = cv2.Laplacian(blurred_image, cv2.CV_64F)
```

```
laplacian_image = cv2.convertScaleAbs(laplacian_image)
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(laplacian_image, cmap='gray')
plt.title('Laplacian of Gaussian (LoG)')
plt.show()
```

4. **Finding 2D Fourier Transform and Plotting It**

Useful URL
This code applies the **2D Fast Fourier Transform (FFT)** to the image using NumPy's fft2() function.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('messi5.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
```

The Fourier Transform result places the **low-frequency components** in the corners of the result. To visualize it better, we use fftshift() to shift these low frequencies to the **center** of the image.

```
magnitude_spectrum = 20*np.log(np.abs(fshift))
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

5. **Some sort of filtering in the frequency domain and then inverse transform and plotting the image**

Useful URL

```
# Reading the Image
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('messi5.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"
```

```
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)


# Create a high-pass filter mask, you can also design low-pass filter or any filter of your choice
rows, cols = img.shape
crow, ccol = rows//2, cols//2
fshift[crow-30:crow+31, ccol-30:ccol+31] = 0
f_ishift = np.fft.ifftshift(fshift)
img_back = np.fft.ifft2(f_ishift)
img_back = np.real(img_back)
plt.subplot(131),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_back, cmap = 'gray')
plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(img_back)
plt.title('Result in JET'), plt.xticks([]), plt.yticks([])
plt.show()
```